Automatic Annotation and Semantic Search from Protégé

Miriam Fernández, David Vallet, and Pablo Castells

Universidad Autónoma de Madrid, Escuela Politécnica Superior
{miriam.fernandez, david.vallet, pablo.castells}@uam.es
http://nets.ii.uam.es

Semantic search has been one of the major envisioned benefits of the Semantic Web since its emergence in the late 90's [1]. Our demo shows a proposal towards this goal. One way to view a semantic search engine is as a tool that gets formal queries (e.g. in RDQL, RQL, SPARQL, or the like) from a client, executes them against an ontology-based knowledge base, and returns tuples of ontology values (resources) that satisfy the query [2]. While this conception of semantic search brings enormous advantages already, our work aims at taking a step beyond this. In our view of Information Retrieval in the Semantic Web, a search engine returns documents, rather than (or in addition to) exact values, in response to user queries. The engine should rank the documents, according to concept-based relevance criteria. The overall retrieval process is illustrated in Figure 1 (see [3] for more details of our research).

In our demo we present an environment where these ideas are put to work. The main ideas behind our prototype, and their realization in the demo, are briefly explained next.

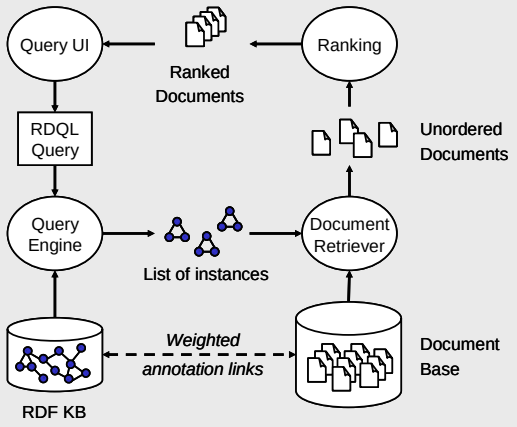

Figure 1. Our view of ontology-based information retrieval

1. Knowledge representation

We propose a model that considers the distinction of three types of ontologies:

- Domain ontologies.
- Content ontologies..
- Topic ontologies.

This model is based on pragmatic considerations, and is materialised through three small root class hierarchies, that ontology engineers can subclass.

- *Concept* should be the root of all domain classes that can be used (directly or after subclassing) to create instances that describe specific entities referred to in the documents.
- *Document* is used to create instances that act as proxies of documents from the information source to be searched upon.
- *Taxonomy* is the root for class hierarchies that are merely used as classification schemes, and are never instantiated.

2. Automatic annotation

The predefined base ontology classes described above are complemented with an annotation ontology that provides the basis for the semantic indexing and ranking of documents.

Documents are annotated with concept instances from the KB by creating instances of the *Annotation* class, provided for this purpose. *Annotation* has two relational properties, *instance* and *document*, by which concepts and documents are related together.

Our system provides a simple facility for semi-automatic annotation, which works as follows. *Concept* instances use a *label* property to store the most usual text form of the concept class or instance. This property is multivalued, since instances may have several textual lexical variants. Instance labels are used by the automatic annotator to find potential occurrences of instances in text documents. Whenever the label of an instance is found, an annotation is created between the instance and the document. Documents can be annotated with classes as well, by assigning labels to concept classes. This basic mechanism is complemented with heuristics to cope with polysemia, i.e. label coincidence between different instances or classes.

Annotation has a *weight* property, used by the retrieval and ranking module. The ranking algorithm is based on an adaptation of the classic vector model. In the classic vector model, keywords appearing in a document are assigned

a relevance weight for the document. Similarly, in our system, annotations are assigned a weight that reflects how relevant the instance is considered to be for the document meaning.

Weights are computed automatically by an adaptation of the IF-TDF algorithm, based on the frequency of occurrence of the instances in each document. Our system provides a separate *keyword* property to be used, in addition to *label*, for instance frequency computation, but not for automatic annotation, in order to avoid polysemic ambiguities that lead to incorrect annotations.

An annotation management environment is provided as a Protégé plugin, which implements and integrates:

- An automatic annotator that links concepts and documents together, as explained above.
- A weighting algorithm that assigns weights to annotations and classifications.
- User interface support to browse through, and edit, the generated weighted annotations.

3. Semantic search

A search tool is available as a Protégé plug-in, which integrates a semantic search engine, a user profile editor, and a query user interface.

The search system retrieves documents in response to formal queries combining two different techniques: keyword-based search and semantic search. In the fist one the query is composed of keywords and the documents containing such keywords are searched for. In the second one the query is entered in RDQL language and the search engine returns all the documents in the repository that are annotated with the ontology instances (and/or classes) that satisfy the RDQL query.

Documents are ranked by an algorithm that adapts the vector-space model principles to an ontology-based representation of document semantics. If the knowledge in the KB is incomplete, the semantic ranking algorithm performs very poorly: RDQL queries will return less results than expected, and the relevant documents will not be retrieved, or will get a much lower similarity value than they should. As limited as might be, keyword-based search could perform better in these cases. Consequently, our ranking model combines the semantic similarity measure with the similarity measure of a keyword-based algorithm. The final value for ranking is computed as $t \times sim(D_i, Q) + (t-1) \times ksim(D_i, Q)$, where *ksim* is computed by the keyword-based algorithm and *sim* is computed by the semantic algorithm.

The user profile editor allows the user to introduce her preferences as a set of concepts of the ontology. These preferences are used in the ranking algorithm to improve the effectiveness of the system, pushing up documents that are relevant for user interests.

The search user interface allows the user to enter queries in RDQL or/and keywords, set advanced query parameters, view and browse the results. The user can tweak the following parameters: the balance between semantic similarity keyword similarity (the *t* parameter above), the degree of personalization, and a weight for each of the variables in a RDQL query.

Search results can be viewed and browsed in two different views: an end-user view, showing a short summary of the returned documents, and a developer view, where the ranking values, document IDs, and other internal details can be inspected and applied separately.

4. Implementation

Our system admits ontologies in both RDF and OWL. The environment has been built using Protégé, along with Jena 2.2, and Jakarta Lucene for the combination and comparison with keyword-based search.

References

1. Kiryakov, A., Popov, B., Terziev, I., Manov, D., Ognyanoff, D.: Semantic Annotation, Indexing, and Retrieval. Journal of Web Sematics 2, Issue 1, Elsevier (2005) 47-49
2. Maedche, A., Staab, S., Stojanovic, N., Studer, R., Sure, Y.: SEmantic portAL: The SEAL Approach. In: Fensel, D., Hendler, J. A., Lieberman, H., Wahlster, W. (eds.): Spinning the Semantic Web. MIT Press, Cambridge London (2003) 317-359.
3. D. Vallet, M. Fernandez, P. Castells. An Ontology-Based Information Retrieval Model. 2nd European Semantic Web Conference (ESWC 2005). Heraklion, Greece, May 2005. Springer Verlag Lecture Notes in Computer Science.